The binding model: Retrieving and validating user input

Keywords:

define a route with parameters

a user requests a given product page

request includes data from a form

using binding models to retrieve those parameters from the request

bind them to C# objects

passed to your Razor Page handlers as method parameters

set as properties on your Razor Page PageModel

page handler executes

values provided by the user are valid

binding models as the input to a Razor Page

populating “plain old CLR objects” (POCOs)

output models in ASP.NET Core’s implementation of MVC

view models and API models

generate a response to the user’s request

recap the MVC design pattern

Questions:

Q: What is the purpose of using binding models in Razor Pages?

A: To retrieve parameters from the request and bind them to C# objects.

Q: How are the bound C# objects used in Razor Pages?

A: They are passed as method parameters or set as properties on the Razor Page PageModel.

Q: Why must you ensure that the values provided by the user are valid?

A: Because the data comes from a user and they can’t be trusted.

Q: What do binding models represent in Razor Pages?

A: The input to a Razor Page, populating “plain old CLR objects” (POCOs).

Q: What are output models used for in ASP.NET Core’s MVC implementation?

A: To generate a response to the user’s request.

Q: Which chapters will cover the output models like view models and API models?

A: Chapters 7 and 9.

Understanding the models in Razor Pages and MVC

Keywords:

binding models fit into the MVC design pattern

separation of concerns

Controller—Calls methods on the model and selects a view

View—Displays a representation of data that makes up the model

Model—The data to display and the methods for updating itself

application model

to-do list application

todo/listcategory/{category}/{username}

ASP.NET Core Razor Pages uses several different models

Binding model—The binding model is all the information that's provided by the user

route parameters parsed from the URL

public property on the page's PageModel

decorating it with the \[BindProperty] attribute

Application model—The application model isn't really an ASP.NET Core model

domain model

database models

Page model—The PageModel of a Razor Page serves two main functions

acts as the controller for the application

acts as the view model for a Razor view

ModelState property

model validation

/search/{query} URL

Binding model—This would take the {query} route parameter from the URL

bind them to a C# class

set as a property on the PageModel

Application model—This is the services and classes that perform the logic

load all the clothes that match the query

Page model—The values provided by the application model would be set as properties on the Razor Page's PageModel

Razor view would use this data to render the Razor view to HTML

responsibilities are well defined and distinct

PageModel, as it is where the binding models and page handlers are defined

holds the data required for rendering the view

binding models that are built from incoming requests

Questions:

Q: What are the three independent components of the classic MVC design pattern?

A: Controller, View, and Model.

Q: What is the role of the Controller in the MVC design pattern?

A: Calls methods on the model and selects a view.

Q: What does the Binding model include in ASP.NET Core Razor Pages?

A: All the information provided by the user such as route parameters, query string, and form or JSON data in the request body.

Q: How are binding models typically defined in Razor Pages?

A: By creating a public property on the page's PageModel and decorating it with the \[BindProperty] attribute.

Q: What does the Application model represent in ASP.NET Core?

A: A group of services and classes needed to perform business actions, including domain and database models.

Q: What two main functions does the PageModel serve in Razor Pages?

A: It acts as the controller by exposing page handler methods and as the view model for a Razor view.

Q: What property on the PageModel contains the result of model validation?

A: The ModelState property.

Q: What does the binding model take from the /search/{query} URL?

A: The {query} route parameter from the URL and any values posted in the request body.

Q: What is the role of the application model in the clothes search application?

A: To load all the clothes that match the query, apply sorting and filters, and return the results to the controller.

Q: What does the page model do with the values from the application model?

A: Sets them as properties on the Razor Page's PageModel along with metadata for rendering the view.

Q: Why is it important to keep the binding, application, and page models separate?

A: To ensure the application stays agile and easy to update by having well-defined and distinct responsibilities.

Q: Where are the binding models and page handlers defined in an ASP.NET Core Razor Pages application?

A: In the PageModel.

Q: What is the next focus after introducing the various models in ASP.NET Core according to the passage?

A: How to use binding models built from incoming requests, including how they are created and where the values come from.

From request to model: Making the request useful

Keywords:

How ASP.NET Core creates binding models from a request

How to bind simple types, like int and string, as well as complex classes

choose which parts of a request are used in the binding model

page handlers are normal C# methods

model binding extracts values from a request

passed as method parameters to the page handler

set as properties of the PageModel that are marked with the \[BindProperty] attribute

model binding in Razor Pages and MVC is a one-way population of objects

only requests using verbs like POST and PUT are bound

set the SupportsGet property on the \[BindProperty] attribute

properties of the request, such as the request URL, any headers

Form values—Sent in the body of an HTTP request

Route values—Obtained from URL segments

Query string values—Passed at the end of the URL

model binder checks each binding source

Id property has been bound from a URL route parameter

Name and SellPrice properties have been bound from the request body

you don’t have to write the code to parse requests

Model binding is great for reducing repetitive code

Questions:

Q: What types of values can ASP.NET Core bind to create binding models?

A: Simple types, like int and string, as well as complex classes.

Q: What does model binding do with values from a request?

A: Extracts values from a request and uses them to create .NET objects.

Q: Where are model-bound objects passed in a Razor Page?

A: As method parameters to the page handler or as properties of the PageModel marked with the \[BindProperty] attribute.

Q: What is the default behavior of model binding for GET requests in Razor Pages?

A: PageModel properties are not model-bound for GET requests, even if you add the \[BindProperty] attribute.

Q: How can you enable model binding for GET requests?

A: By setting the SupportsGet property on the \[BindProperty] attribute.

Q: What request sources does ASP.NET Core use for model binding?

A: Form values, route values, and query string values.

Q: What advantage does model binding provide over manually parsing requests?

A: You don’t have to write the code to parse requests and map the data yourself.

Q: What is model binding in Razor Pages and MVC described as?

A: A one-way population of objects from the request.

Q: What does the model binder do with each binding source?

A: Checks each binding source to see if it contains a value that could be set on the model.

Q: Which properties were bound from the request body in the example given?

A: The Name and SellPrice properties.

Q: Why is it beneficial to use model binding in ASP.NET Core?

A: It reduces repetitive code and lets you focus on business requirements.

Q: How often do developers need to customize the way model binding works?

A: It’s relatively rare that you’ll find yourself needing to dig too deep into this.

Binding simple types

Keywords:

simple Razor Page handler

takes one number as a method parameter

squares it by multiplying the number by itself

uses routing to parse it for route parameters

number=5

OnGet page handler contains a single parameter

binding model

spot the expected parameter

flick through the route values

bind the number parameter

you didn’t have to write any extra code

model binding do its magic

three default binding sources

Form values

Route values

Query string values

store values as name-value pairs

set to a new, default instance of the type

default(T)

parameterless constructor

nullable types like int? or bool?

string types, the value will be null

model binding fails to bind your method parameters

value passed to the method could be null

bind a single method parameter

bind multiple method parameters

customize the route template for the page

accepts the three values you need

Questions:

Q: What does the Razor Page handler do when a client requests /CalculateSquare/5?

A: It uses routing to parse it for route parameters.

Q: What value is produced from the route when accessing /CalculateSquare/5?

A: number=5

Q: What is the binding model in the OnGet page handler?

A: An integer called a number.

Q: How does ASP.NET Core bind the number parameter in the handler method?

A: It finds the number=5 pair in the route values and binds it.

Q: What coding effort is required to extract the number from the URL?

A: You didn't have to write any extra code.

Q: What are the three default binding sources used by the model binder?

A: Form values, route values, query string values.

Q: How are values stored in each of the binding sources?

A: As name-value pairs.

Q: What happens if none of the binding sources contain the required value?

A: The binding model is set to a new, default instance of the type.

Q: What is the default value of an int parameter if binding fails?

A: 0

Q: What value is assigned to a string type if binding fails?

A: null

Q: What is the result for nullable types like int? or bool? when no value is found?

A: The value will be null.

Q: Why is it important to consider behavior when model binding fails?

A: The value passed to the method could be null or could unexpectedly have a default value.

Q: What does Listing 6.2 demonstrate?

A: How to bind a single method parameter.

Q: What is the next logical step after binding a single method parameter?

A: Bind multiple method parameters.

Q: How do you customize the route template for Convert.cshtml?

A: By using the @page directive to use an absolute path containing two route values.

Q: What does the new page handler in Convert.cshtml accept?

A: The three values you need.

A Razor Page handler accepting multiple binding parameters

Keywords:

three different parameters to bind

depending on the data sent

different values will be bound

available binding sources offer conflicting values

qty value isn't found

default value of 0

duplicated values

form values will take precedence over other binding sources

default model binder isn't case sensitive

QTY=50 will happily bind to the qty parameter

relatively unusual to be binding from all these different sources at once

request body as form values

ID from URL route values

cautionary tale

values stored in binding sources are all strings

convert a string to

primitive .NET type

anything that has a TypeConverter

Type

Questions:

Q: What does Table 6.1 show about model binding?

A: A whole variety of possibilities.

Q: What causes different values to be bound in the same handler?

A: Depending on the data sent, different values will be bound.

Q: What happens when the qty value isn't found in form data, route values, or query string?

A: It has the default value of 0.

Q: In cases of duplicated values, what must be kept in mind?

A: The order in which the model binder consults the binding sources.

Q: Which binding source takes precedence over others by default?

A: Form values.

Q: Is the default model binder case sensitive?

A: The default model binder isn't case sensitive.

Q: What will happen if a binding value of QTY=50 is provided?

A: It will happily bind to the qty parameter.

Q: How common is it to bind from multiple sources at once?

A: It's relatively unusual to be binding from all these different sources at once.

Q: What is a more common scenario for binding?

A: Values all come from the request body as form values, maybe with an ID from URL route values.

Q: What type are values stored in binding sources?

A: Strings.

Q: What types can the model binder convert a string to?

A: Any primitive .NET type such as int, float, decimal, plus anything that has a TypeConverter.

Q: What is an example of a special case that can be converted from a string?

A: Type.

Binding complex types

Keywords:

only being able to bind simple primitive types

convert strings directly to those primitive types

bind complex types by traversing any properties your binding models expose

page handlers if simple types were your only option

currency converter application

exchange some currency

collect their name, email, and phone number

public IActionResult OnPost(string firstName, string lastName, string phoneNumber, string email)

method signature will keep growing

not exactly clean code

\[BindProperty] approach

clutter up our PageModel with lots of properties and attributes

Questions:

Q: What is a limitation if the model binder could only bind simple primitive types?

A: Only being able to bind simple primitive types is a bit limiting.

Q: What can the model binder do beyond converting strings to primitive types?

A: Bind complex types by traversing any properties your binding models expose.

Q: What kind of method would be needed if simple types were the only option?

A: A method like public IActionResult OnPost(string firstName, string lastName, string phoneNumber, string email).

Q: What happens when requirements change and more details need to be collected?

A: The method signature will keep growing.

Q: How does the model binder behave when binding multiple simple parameters?

A: The model binder will bind the values quite happily.

Q: Why is using many simple parameters not considered clean code?

A: It’s not exactly clean code.

Q: Why doesn't using the \[BindProperty] approach solve the clutter issue?

A: You still have to clutter up our PageModel with lots of properties and attributes.

SIMPLIFYING METHOD PARAMETERS BY BINDING TO COMPLEX OBJECTS

(A binding model for capturing a user's details)

Keywords:

extract a class that encapsulates the data the method requires

this class becomes your binding model

model binder treats this new complex type a little differently

new instance of the model using new UserBindingModel()

bind to the simple parameter

loops through all the properties your binding model has

consults the collection of binding sources

properties prefixed with the name of the property

multiple complex parameters to a page handler

avoid this situation if possible

casing of the prefix does not matter

model is passed to the page handler

code is cleaner and easier to work with

must have a default public constructor

only bind properties that are public and settable

bind complex hierarchical models

binder can traverse it with ease

Questions:

Q: What is a common pattern in C# when a method has many parameters?

A: Extract a class that encapsulates the data the method requires.

Q: What does this class become when used for model binding?

A: This class becomes your binding model.

Q: How does the model binder treat complex types differently from simple types?

A: The model binder creates a new instance of the model using new UserBindingModel().

Q: When is it more appropriate to use a simple parameter instead of a custom class?

A: If your page handler needs only a single integer, then it makes more sense to bind to the simple parameter.

Q: What does the model binder do with each property of the binding model?

A: It consults the collection of binding sources and attempts to find a name-value pair that matches.

Q: What additional naming pattern does the model binder look for in properties?

A: Properties prefixed with the name of the property, such as user.FirstName and user.LastName.

Q: What should be avoided when using multiple complex parameters?

A: You should avoid this situation if possible.

Q: What happens once the binding model's properties are set?

A: The model is passed to the page handler (or the \[BindProperty] property is set).

Q: What requirement must a class meet to be model-bound?

A: It must have a default public constructor.

Q: What types of properties can be bound in a model-bound class?

A: Only bind properties that are public and settable.

Q: What type of models can be bound using this technique?

A: Complex hierarchical models whose properties are themselves complex models.

Q: How does the binder handle nested properties in a complex model?

A: The binder can traverse it with ease.

BINDING COLLECTIONS AND DICTIONARIES

Keywords:

bind to collections, lists, and dictionaries

user selected all the currencies they were interested in

create a page handler that accepts a List<string> type

currencies\[index]

you can omit the name of the parameter

omit the index and send currencies as the key for every value

key values can come from route values and query values

POST them in a form

Dictionaries can use similar binding

the dictionary key replaces the index

don’t feel too alarmed

Razor views to generate HTML

take care of generating the correct names

form data you POST will be generated in the correct format

Questions:

Q: What types can you bind to in addition to ordinary custom classes and primitives?

A: You can bind to collections, lists, and dictionaries.

Q: What does the example page display based on user selection?

A: The rates for all those selected.

Q: What type is accepted by the example page handler for currency selection?

A: A List<string> type.

Q: What format uses the parameter name and index to send data?

A: currencies\[index].

Q: What format omits the parameter name for a single list?

A: \[index].

Q: What format omits the index and repeats the key for each value?

A: currencies=GBP\&currencies=USD.

Q: Where do the key values commonly come from?

A: It’s far more common to POST them in a form.

Q: What replaces the index when binding to a dictionary?

A: The dictionary key replaces the index.

Q: What helps avoid confusion when generating form data in traditional web apps?

A: Razor views to generate HTML.

Q: Who takes care of generating the correct names for POSTed form data?

A: The framework.

Q: What will Razor views ensure about the form data you POST?

A: It will be generated in the correct format.

BINDING FILE UPLOADS WITH IFORMFILE

Keywords:

upload files

IFormFile interface

binding model

\[BindProperty] approach

IEnumerable<IFormFile>

exposes several properties and utility methods

FileName property

never use the filename directly in your code

whole content of the file is buffered in memory and on disk

OpenReadStream method

stream the files directly

streaming large files can be complex and error-prone

Don’t use the IFormFile interface to handle large file uploads

avoid file uploads entirely

default configuration of model binding

override the process

replacing the ModelBinders

specify which binding source to use

Questions:

Q: What common feature of many websites is discussed in the passage?

A: The ability to upload files.

Q: What interface does ASP.NET Core use to support file uploads?

A: IFormFile interface.

Q: How can the IFormFile interface be used in a page handler?

A: As a method parameter or using the \[BindProperty] approach.

Q: What should you use if you need to accept multiple files?

A: IEnumerable<IFormFile>.

Q: What does the FileName property of IFormFile return?

A: The filename that the file was uploaded with.

Q: What warning is given about using posted filenames in code?

A: Never use posted filenames in your code.

Q: What happens when a method accepts an IFormFile instance?

A: The whole content of the file is buffered in memory and on disk.

Q: What method is used to read data from an IFormFile object?

A: OpenReadStream.

Q: What problem might occur if users post large files using IFormFile?

A: You may start to run out of space in memory or on disk.

Q: What is said about streaming large files?

A: Streaming large files can be complex and error-prone.

Q: What is recommended instead of handling large file uploads?

A: Avoid file uploads entirely.

Q: What can you do if you need more control over model binding?

A: Completely override the process by replacing the ModelBinders.

Q: What is more common than customizing the entire binding process?

A: Specify which binding source to use for a page's binding model.

Choosing a binding source

Keywords:

bind your binding models from three different binding sources

form data, route data, and the query string

specifically declare which binding source to bind to

bind a method parameter to a request header value

body of a request contains JSON-formatted data

\[FromHeader] attribute

bind the value to an HTTP request header value

\[FromBody] attribute

read JSON from the body of the request

\[FromBody] attribute is explicitly required

only a JSON input formatter is configured

\[FromQuery]—Bind to a query string value

\[FromRoute]—Bind to route parameters

\[FromForm]—Bind to form data

only one value may be decorated with the \[FromBody] attribute

\[BindNever]—The model binder will skip this parameter

\[BindRequired]—the binder will add a validation error

\[FromServices]—provided using dependency injection

\[ModelBinder] attribute

configure the model binder to bind to pretty much any request data

validate the input coming in

declarative manner out of the box

Questions:

Q: What three binding sources does the ASP.NET Core model binder use by default?

A: Form data, route data, and the query string.

Q: When might you need to declare a specific binding source?

A: When you want to bind a method parameter to a request header value or bind JSON-formatted data in the request body.

Q: What attribute binds a method parameter to an HTTP request header value?

A: \[FromHeader] attribute.

Q: What does the \[FromBody] attribute do in a page handler?

A: It binds JSON from the body of the request to a method parameter.

Q: What warning is given about the \[FromBody] attribute in Razor Pages?

A: The \[FromBody] attribute is explicitly required when binding to JSON requests in Razor Pages.

Q: What format is configured by default for binding data from the request body?

A: Only a JSON input formatter is configured.

Q: What are the five attributes that override default binding sources?

A: \[FromHeader], \[FromQuery], \[FromRoute], \[FromForm], and \[FromBody].

Q: What is the limitation of using the \[FromBody] attribute?

A: Only one parameter may use the \[FromBody] attribute.

Q: Why can only one parameter use the \[FromBody] attribute?

A: HTTP request bodies can only be safely read once.

Q: What attribute prevents the model binder from binding a parameter?

A: \[BindNever].

Q: What happens if a parameter marked with \[BindRequired] is not provided?

A: The binder will add a validation error.

Q: What does the \[FromServices] attribute indicate?

A: The parameter should be provided using dependency injection.

Q: What can the \[ModelBinder] attribute do?

A: It allows you to specify the exact binding source, override the name, and specify the type of binding.

Q: How can you configure the model binder to bind to request data?

A: By combining all these attributes.

Q: Why is validating incoming data important?

A: Because the binder is blindly assigning values and there's nothing stopping malicious data from being sent.

Q: What does ASP.NET Core provide to help validate input data?

A: A declarative manner out of the box.

Handling user input with model validation

Keywords:

What is validation, and why do you need it?

Using DataAnnotations attributes to describe the data you expect

validate your binding models in page handlers

relatively easy to add validation

you definitely shouldn't trust the data sent as part of a request

model validation fits in this process

validate data provided by users before you use it

check for non-malicious errors

email fields have a valid email format

can't buy -1 copies of this book

name may be required for a profile but phone number is optional

can't convert a currency to itself

client-side validation is useful for users

always necessary to validate the data

validate user input on the server side

duplication is a necessary evil

Blazor, the new C# SPA framework

simplify your validation code significantly

attributes provided by .NET Core and .NET 5.0

Questions:

Q: What is the main purpose of validation in ASP.NET Core applications?

A: To ensure the data you receive is safe and conforms to expectations before using it in your methods.

Q: When does validation occur in the Razor Pages framework?

A: After model binding, but before the page handler executes.

Q: What kind of data should always be validated before being used?

A: Data provided by users.

Q: What is the classic example used to emphasize the importance of validating user data?

A: Little Bobby Tables.

Q: What are some examples of non-malicious data errors?

A: Email fields have a valid email format, numbers might need to be in a particular range, some values may be required, and values must conform to business requirements.

Q: Why can’t developers rely only on client-side validation?

A: Because it will always be possible to bypass these browser protections.

Q: What kind of validation is always necessary for incoming data?

A: Server-side validation.

Q: What does the passage say about duplicating validation logic?

A: The duplication is a necessary evil.

Q: What does ASP.NET Core provide to reduce the burden of writing validation code?

A: Several features to try to reduce this burden.

Q: What framework is mentioned as a potential solution to some validation issues?

A: Blazor, the new C# SPA framework.

Q: What can simplify your validation code significantly?

A: A set of attributes provided by .NET Core and .NET 5.0.

Using DataAnnotations attributes for validation

Keywords:

Validation attributes, or more precisely DataAnnotations attributes

specify the rules that your binding model should conform to

metadata about your model

DataAnnotations attributes directly to your binding models

check that required fields have been provided

valid email addresses

UserBindingModel decorated with validation attributes

whole wealth of information

FirstName property should always be provided

maximum length of 100 characters

clearly declare the expected state of the model

hooks for the ASP.NET Core framework to validate

plethora of attributes to choose from

System.ComponentModel.DataAnnotations namespace

\[EmailAddress]—Validates that a property has a valid email address format

\[Required]—Indicates the property must not be null

\[Compare]—Allows you to confirm that two properties have the same value

only validate that the format of the value is correct

been part of the .NET Framework since version 3.5

used for other purposes, in addition to validation

define the types of columns and rules

write custom attributes by deriving from the base ValidationAttribute

FluentValidation library

not so good for validating business rules

validation occurs, but it doesn't automatically do anything if validation fails

Questions:

Q: What do DataAnnotations attributes allow you to specify?

A: The rules that your binding model should conform to.

Q: What is metadata described as in the passage?

A: Metadata describes other data, specifying the rules and characteristics the data should adhere to.

Q: How can you indicate acceptable data types for your binding models?

A: By applying DataAnnotations attributes directly to your binding models.

Q: What example is used to show a binding model with validation attributes?

A: The checkout page for your currency converter application.

Q: What is one rule applied to the FirstName property in the UserBindingModel?

A: It should always be provided.

Q: What does the ASP.NET Core framework use attributes for during model binding?

A: To validate that the data set on the model is valid.

Q: What namespace contains more DataAnnotations attributes?

A: System.ComponentModel.DataAnnotations namespace.

Q: What does the \[EmailAddress] attribute validate?

A: That a property has a valid email address format.

Q: What does the \[Required] attribute indicate?

A: The property must not be null.

Q: What does the \[Compare] attribute allow you to do?

A: Confirm that two properties have the same value.

Q: What is a limitation of attributes like \[EmailAddress]?

A: They only validate that the format of the value is correct.

Q: Since when have DataAnnotations attributes been part of the .NET Framework?

A: Since version 3.5.

Q: What other purpose do DataAnnotations attributes serve besides validation?

A: They define the types of columns and rules when creating database tables from C# classes.

Q: How can you create custom validation logic if built-in attributes are insufficient?

A: By deriving from the base ValidationAttribute.

Q: What library can be used instead of DataAnnotations attributes?

A: FluentValidation library.

Q: Why are DataAnnotations not ideal for validating business rules?

A: Because they are good for input validation of properties in isolation, but not so good for validating business rules.

Q: What does the Razor Pages framework ensure regarding validation?

A: That validation occurs.

Q: What does the framework not do automatically when validation fails?

A: It doesn’t automatically do anything if validation fails.

Validating on the server for safety

Keywords:

Validation of the binding model occurs before the page handler executes

the handler always executes, whether the validation failed or succeeded

responsibility of the page handler to check the result of the validation

Validation happens automatically

handling validation failures is the responsibility of the page handler

Razor Pages framework stores the output of the validation attempt

ModelState property on the PageModel

ModelStateDictionary object

contains a list of all the validation errors

OnPost page handler for the Checkout.cshtml Razor Page

Input property is marked for binding

UserBindingModel type

Questions:

Q: When does validation of the binding model occur in relation to the page handler execution?

A: Validation of the binding model occurs before the page handler executes.

Q: Does the page handler execute if validation fails?

A: Yes, the handler always executes, whether the validation failed or succeeded.

Q: Whose responsibility is it to check the result of the validation?

A: It is the responsibility of the page handler to check the result of the validation.

Q: Where does the Razor Pages framework store the output of the validation attempt?

A: It stores the output in a property on the PageModel called ModelState.

Q: What type of object is the ModelState property?

A: ModelState is a ModelStateDictionary object.

Q: What does the ModelStateDictionary contain?

A: It contains a list of all the validation errors that occurred after model binding.

Q: What is shown in the example provided in the passage?

A: The OnPost page handler for the Checkout.cshtml Razor Page.

Q: What is notable about the Input property in the example?

A: The Input property is marked for binding and uses the UserBindingModel type.

Checking model state to view the validation result

Keywords:

ModelState property indicates an error occurred

calls the Page helper method

returns a PageResult that will ultimately generate HTML

view uses the (invalid) values provided in the Input property

helpful messages for the user are added automatically

error messages displayed on the form are the default values for each validation attribute

customize the message by setting the ErrorMessage property

page handler returns a RedirectToPageResult

POST-REDIRECT-GET pattern

generation of the validation check and response automatic

Razor Pages apps typically still want to generate an HTML response

loading a list of available currencies

IsValid pattern

DataAnnotations validation won't know whether a product with the requested ID exists

treat data and business rule validation failures in the same way

only performing validation on the server can leave users with a slightly poor experience

have that feedback immediately

Questions:

Q: What happens if the ModelState property indicates an error occurred?

A: The method immediately calls the Page helper method, returning a PageResult that generates HTML to return to the user.

Q: How are error messages on the form generated and can they be customized?

A: They are the default values for each validation attribute but can be customized by setting the ErrorMessage property on any validation attribute.

Q: What does the page handler return if the request is successful?

A: It returns a RedirectToPageResult that redirects the user to the Success.cshtml Razor Page.

Q: What is the POST-REDIRECT-GET pattern?

A: It is the pattern of returning a redirect response after a successful POST.

Q: Why doesn't ASP.NET Core automatically handle invalid requests for Razor Pages?

A: Because Razor Pages typically still want to generate an HTML response to show the user the problem and allow corrections.

Q: What benefit does the IsValid pattern provide in page handlers?

A: It makes loading additional data simpler and explicit and helps control what happens when validation checks fail.

Q: Why might you move validation checks to the handler method?

A: To treat data and business rule validation failures in the same way.

Q: What user experience issue does only performing validation on the server cause?

A: Users may submit a form, leave, and return to find errors, which requires redoing the form.

Q: What is the suggested improvement to enhance user experience during validation?

A: Providing immediate feedback on form errors.

Validating on the client for user experience

Keywords:

client-side validation

HTML5 has several built-in validation behaviors

“email” HTML input type

browser will automatically stop you from submitting an invalid format

perform client-side validation by running JavaScript

most common approach used in Razor Pages

DataAnnotations attributes

decorate a view model with these attributes

user sees any errors with their form immediately

shorter feedback cycle

building an SPA

client-side framework to validate the data

Web API will still validate the data when it arrives at the server

Razor Pages to generate your HTML

automatically configures client-side validation for most of the built-in attributes

custom ValidationAttributes will only run on the server by default

need to do some additional wiring up

model binding framework in ASP.NET Core

organize your Razor Pages

Questions:

Q: What built-in feature in HTML5 helps stop submission of invalid email formats?

A: The “email” HTML input type causes the browser to automatically stop submission of invalid formats.

Q: What is the most common approach used in Razor Pages for client-side validation?

A: Performing client-side validation by running JavaScript on the page before submitting the form.

Q: How do DataAnnotations attributes help with client-side validation?

A: By decorating a view model with these attributes, they provide metadata for Razor to generate the appropriate HTML for validation.

Q: What advantage does client-side validation provide to users?

A: Users see any errors with their form immediately, providing a much shorter feedback cycle and better user experience.

Q: In an SPA, who is responsible for client-side validation?

A: The client-side framework is responsible for validating data before posting it to the Web API.

Q: How does Razor Pages support client-side validation automatically?

A: It automatically configures client-side validation for most of the built-in attributes without requiring additional work.

Q: What limitation exists for custom ValidationAttributes in Razor Pages?

A: Custom ValidationAttributes will only run on the server by default and require additional wiring to work on the client side.

Q: What flexibility does the ASP.NET Core model binding framework offer?

A: It offers many options for organizing Razor Pages, such as using page handler parameters or PageModel properties, and defining binding model classes in different ways.

Organizing your binding models in Razor Pages

(Designing an edit product Razor Page)

Keywords:

general advice on how I like to configure the binding models

your Razor Pages will follow a consistent layout

personal preference

Model binding in ASP.NET Core has a lot of equivalent approaches to take

example of how I would design a simple Razor Page

PageModel for a typical “edit form”

model-binding approach

Only bind a single property with \[BindProperty]

create a separate class, InputModel, to hold the values

Define your binding model as a nested class

InputModel as a nested class inside my Razor Page

Don’t use \[BindProperties]

cause all properties in your model to be model-bound

open to overposting attacks

Accept route parameters in the page handler

avoid the clunky Supports-Get=true syntax

Always validate before using data

use of DataAnnotations can make it easy to add validation to your models

generate HTML in response to a request using the Razor templating engine

Questions:

Q: What is the purpose of following the patterns suggested for configuring binding models in Razor Pages?

A: To make your Razor Pages follow a consistent layout, making it easier for others to understand how each Razor Page works.

Q: What is the author’s stance on the guidelines provided for binding models?

A: The advice is personal preference, and you can adapt it if you don’t agree with some aspects.

Q: What is the recommended approach for binding properties in Razor Pages according to the passage?

A: Only bind a single property with \[BindProperty], and if more values need binding, use a separate InputModel class decorated with \[BindProperty].

Q: Why does the author prefer defining the binding model as a nested class?

A: Because the binding model is normally highly specific to that single page, keeping everything together and adding consistency.

Q: Why does the author suggest not using the \[BindProperties] attribute?

A: Because it binds all properties in the model, which can leave you open to overposting attacks.

Q: How does the author recommend handling route parameters in page handlers?

A: By adding parameters directly to the page handler method itself to avoid the Supports-Get=true syntax.

Q: What is emphasized as an important practice before using data in Razor Pages?

A: Always validate user input.

Q: What role do DataAnnotations play in Razor Pages model binding?

A: They make it easy to add validation to your models.

Q: What topic will the next chapter cover according to the passage?

A: How to create views and generate HTML using the Razor templating engine.